**Omar Amr Adel**

**220357**

**Assignment problem Algorithm**

This code is an implementation of a Matrix Calculation GUI using the Tkinter library in Python. It allows users to input a matrix, perform calculations on it and solve it using assignment problem algorithm, and displays the results.

**Code breakdown:**

import tkinter as tk

from PIL import ImageTk, Image

from tkinter import ttk

Importing libraries:

tkinter is imported as tk for creating the GUI components and tkk for styling.

ImageTk and Image from the PIL library are imported for handling image

**Function Definitions:**

def get\_matrix\_size():

    size = int(size\_entry.get())

    return size

get\_matrix\_size() retrieves the size of the matrix from the size\_entry input field.

def sum\_zero\_cells(original\_matrix, result\_matrix):

    size = len(original\_matrix)

    total\_cost = 0

    for i in range(size):

        j = size -1

        while True:

            if result\_matrix[i][j] == 0:

                original\_value = original\_matrix[i][j]

                total\_cost += original\_value

                break  # Stop at the first zero in each row

            j -= 1

    return total\_cost

sum\_zero\_cells() calculates the total cost by summing the values of zero cells in the result matrix.

def sum\_zero\_cells\_Details(original\_matrix, result\_matrix):

    size = len(original\_matrix)

    total\_cost = []

    for i in range(size):

        j = size -1

        while True:

            if result\_matrix[i][j] == 0:

                original\_value = original\_matrix[i][j]

                total\_cost.append(original\_value)

                total\_cost.append("+")

                break  # Stop at the first zero in each row

            j -= 1

    return total\_cost

sum\_zero\_cells\_Details() provides a detailed breakdown of the zero cell values and the addition operations.

def get\_matrix\_values(size):

    matrix = []

    for i in range(size):

        row = []

        for j in range(size):

            value = int(matrix\_entries[i][j].get())

            row.append(value)

        matrix.append(row)

    return matrix

get\_matrix\_values() retrieves the values of the matrix from the input fields.

def print\_matrix(matrix):

    output\_text.insert(tk.END, '\n'.join([' '.join(map(str, row)) for row in matrix]) + '\n')

    output\_text.insert(tk.END, "============================" '\n')

def get\_row\_min(row):

    return min(row)

def get\_column\_min(matrix, col):

    return min([row[col] for row in matrix])

def subtract\_row(matrix, row, value):

    new\_row = [cell - value for cell in row]

    matrix[matrix.index(row)] = new\_row

def subtract\_column(matrix, col, value):

    for row in matrix:

        row[col] -= value

print\_matrix() displays a matrix in the output text area.

get\_row\_min() retrieves the minimum value in a row.

get\_column\_min() retrieves the minimum value in a column.

subtract\_row() subtracts a value from each element in a row.

subtract\_column() subtracts a value from each element in a column.

def subtract\_smallest\_top\_right(matrix):

    size = len(matrix)

    flag = 0

    for row in matrix[2:]:

        for element in row:

            if element != 0:

                flag = 1

    for row in matrix:

        for element in row[:-2]:

            if element != 0:

                flag = 1

    if flag == 0 and matrix[0][-1] != 0  and matrix[1][-1] != 0 and matrix[0][-2] != 0 and matrix[1][-2] != 0 and size == 3:

        smallest = min(matrix[0][-1], matrix[1][-1], matrix[0][-2], matrix[1][-2])

        matrix[0][-1] -= smallest

        matrix[1][-1] -= smallest

        matrix[0][-2] -= smallest

        matrix[1][-2] -= smallest

    return matrix

subtract\_smallest\_top\_right() subtracts the smallest value from the top-right section of the matrix.

def calculate\_assignment():

    size = get\_matrix\_size()

    original\_matrix = get\_matrix\_values(size)

    original\_matrix\_copy = copy\_matrix(original\_matrix)

    # First row substitution

    for row in original\_matrix:

        min\_value = get\_row\_min(row)

        subtract\_row(original\_matrix, row, min\_value)

    print\_matrix(original\_matrix)

    # Column substitution

    for i in range(size):

        col\_min\_value = get\_column\_min(original\_matrix, i)

        subtract\_column(original\_matrix, i, col\_min\_value)

    print\_matrix(original\_matrix)

    # Last iteration

    max\_iterations = 1000

    counter = 0

    while True:

        zeros = []

        for i in range(size):

            for j in range(size):

                if original\_matrix[i][j] == 0:

                    zeros.append((i, j))

        if len(zeros) == size \* size:

            break

        else:

            rows = [zero[0] for zero in zeros]

            cols = [zero[1] for zero in zeros]

            min\_value = min([original\_matrix[row][col] for row in rows for col in cols])

            for zero in zeros:

                original\_matrix[zero[0]][zero[1]] -= min\_value

        counter += 1

        if counter > max\_iterations:

            break

    print\_matrix(original\_matrix)

    result\_matrix = subtract\_smallest\_top\_right(original\_matrix)

    print("Updated result matrix:")

    print\_matrix(result\_matrix)

    total\_cost = sum\_zero\_cells(original\_matrix\_copy, result\_matrix)

    total\_cost2 = sum\_zero\_cells\_Details(original\_matrix\_copy, result\_matrix)

    print("Total cost: ", end="")

    print(" =", total\_cost)

    # Display result in GUI

    output\_text.insert(tk.END, "\nUpdated result matrix:\n")

    print\_matrix(result\_matrix)

    print\_cost(total\_cost2)

    output\_text.insert(tk.END, "\nTotal cost: " + str(total\_cost) + "\n")

calculate\_assignment() performs the matrix calculation, including row and column substitutions and finding the assignment.

def copy\_matrix(source\_matrix):

    size = len(source\_matrix)

    Copy = [[0] \* size for \_ in range(size)]  # Create a matrix of zeros with the same size as the source matrix

    for i in range(size):

        for j in range(size):

            Copy[i][j] = source\_matrix[i][j]  # Copy the value from the source matrix to the Copy matrix

    return Copy

def print\_cost(cost):

    cost\_str = ' '.join(map(str, cost[:-1]))

copy\_matrix() creates a copy of the source matrix.

print\_cost() displays the cost breakdown in the output text area.

generate\_button = tk.Button(window, text="Generate Matrix", command=generate\_matrix\_inputs)

generate\_button.pack(pady=10)

# Calculate Button

calculate\_button = tk.Button(window, text="Calculate", command=calculate\_assignment)

calculate\_button.pack(pady=10)

# Output Text Area

output\_text = tk.Text(window, width=40, height=10)

output\_text.pack(pady=10)

exit\_button = tk.Button(window, text="Exit", command=exit\_program)

Generating necessary buttons for GUI.

**GUI Creation:**

The Tkinter Tk class is used to create the main window.

The background image is loaded, resized, and displayed using a Label widget.

The "Matrix Size" label and entry field are created.

The "Generate Matrix" button creates the matrix input fields based on the specified size.

The "Calculate" button triggers the calculate\_assignment() function.

The output text area is created using a Text widget.

The "Exit" button is used to close the program.

**Main Loop:**

The window.mainloop() function is called to start the GUI event loop.